Discovering Relationships between Data Structures and Algorithms

Guojin Zhu
Dept. of Computer Science, Donghua University, Shanghai 201620, China
Email: gjzhu.dhu@163.com
Zhiyue Yu and Jiyun Li
Dept. of Computer Science, Donghua University, Shanghai 201620, China
Email: {yzy8701@163.com, jyli@dhu.edu.cn}

Abstract—There are numerous of program code resources on the web which are solutions to programming problems on online judges. These program code resources are not organized for students to learn data structures and algorithms although they contain much knowledge of data structures and algorithms. For this reason, we propose an approach to organize the program code resources together with the programming problems systematically in terms of algorithms and data structures. This approach is based on the discovery of associate relationships between data structures and algorithms by applying ontology techniques. 1073 program codes on the web which are solutions to 480 problems distributed on online judges were mined in our experiment to discover the relationships between the data structures and algorithms used in the program codes. With the discovered relationships, the program codes and the corresponding problems were organized into learning materials in terms of algorithms and data structures. We believe that it would be useful for students to learn the programming knowledge.

Index Terms—Data structure, Online Judge, Program code, Programming problem, Ontology

I. INTRODUCTION

Nowadays, there are a lot of solution reports of programming problems on the web in the form of blogs written by programming contestants [1]. Most of these reports contain program source codes that can solve the programming problems on online judge (OJ) systems which are gathered from the ACM International Collegiate Programming Contest (ACM/ICPC) [2]. These program code resources contain lots of programming knowledge including data structures and algorithms. As we know, data structures and algorithms play important roles in programming. When students learn programming, they may not only want to learn the knowledge of data structures and algorithms, but also hope to find out suitable programming materials to learn how to implement data structures and algorithms in solving problems. However, the program code resources on the web are not organized in terms of data structures and algorithms. It is difficult for students to find suitable solution reports of programming problems which are focused on the knowledge points of algorithms and data structures that are just taught in their class.

Some approaches have been proposed to organize the program code resources and programming problems. One method is to discover knowledge units for programming tutoring based on Formal Concept Analysis (FCA) [3]. It organizes programming problems by analyzing the source codes submitted by students into a sequence of knowledge units, each of which consists of problems whose solutions need a common group of programming language points. It just deals with the programming knowledge at the level of a programming language, but does not analyze the knowledge of data structures and algorithms. This method is hardly applied to organizing the programming problems on the web into learning materials. Another method is to use a search engine to obtain solution reports on the web and organize them on a basis of a predefined hierarchical body of programming knowledge [1]. Although this method could connect solution reports on the web together, it does not organize the program code resources in terms of algorithms and data structures systematically.

To address the issue above, we propose an approach to organize the program code resources together with the programming problems systematically in terms of algorithms and data structures. We use ontology techniques to recognize the data structures and algorithms used in program codes, and discover the relationships between them. With the discovered relationships, the program codes on the web and the corresponding problems on online judges are organized into learning materials in terms of algorithms and data structures. Furthermore, the programming problems are sorted from easy to difficult.

The main steps of our approach are shown in Fig. 1. First, we construct ontology individuals for program codes on a basis of a knowledge base. Then, we apply reasoning rules to the constructed ontology individuals for recognition of the data structures and algorithms used in the program codes. After that, we discover the relationships between data structures and algorithms by
analyzing the statistical frequencies that each pair of a data structure and an algorithm is recognized appearing in the same program codes. And finally, with hyperlink techniques, we organize the program codes on the web and the corresponding problems on online judges into learning materials for students to learn the programming knowledge in terms of algorithms and data structures.

The rest of this paper is organized as follows. In Section II, we briefly introduce the concept of ontology and construct a knowledge base about data structures and algorithms. In Section III, we recognize the data structures and algorithms used in program codes, and discover the relationships between data structures and algorithms lying in program codes. In Section IV, we introduce the programming problems distributed on online judges and organize them into learning materials in terms of algorithms and data structures. Section V shows the experiment results and the hierarchical structure of the learning materials. Finally, we conclude our paper.

II. KNOWLEDGE BASE

A knowledge base [4] is a special kind of database for knowledge management. Knowledge representation is the core of a knowledge base. Since ontology provides a clear semantic and knowledge description of concepts and interrelation [5], the ontology representation is one of the common methods to represent knowledge.

In order to recognize data structures and algorithms used in program codes, we construct a knowledge base by ontology techniques. This knowledge base mainly contains two types of knowledge. One type is the descriptive knowledge about data structures and algorithms, and the other is the inferential knowledge in the form of reasoning rules which would be used for recognizing the data structures and algorithms contained in program codes.

A. Signals of Data Structures and Algorithms

We give a definition that the signals of data structures and algorithms are strings which can help us to recognize the data structures and algorithms used in program codes. For a data structure, a signal refers to a string that represents its name (or alias) or the name (or alias) of one of its typical operations. For example, the data structure Stack may have the following signals: its name or alias such as mystack or mysta, the name or alias of its pop operation like pop or pop_stack and the name or alias of its push operation like push or push_stack. For algorithms, we regard their names or aliases, which are usually used by programmers, as their signals. Take the algorithm Depth First Search as an example, its signals may be as following strings: depth_first_search, depthfirstsearch, dfs, depthsearch, depth_search, depthfirst and depth_first.

B. Ontology for the Knowledge Base

As a powerful tool, ontology has been widely applied in social science, medicine science and computer science [6]. In the field of computer science, the definition of ontology is that it is a clear formal specification of a shared conceptual model. This definition illustrates four characteristics of ontology [7]: clarification, conceptualization, formalization, and sharing. These features of ontology make it suitable for knowledge representation. The ontology representation [8] is one of commonly used methods to represent knowledge. Ontology can be applied to extract information from texts and documents [9-11]. It can also be used to retrieve information in some other fields, such as e-commerce [12] and crop diseases [13].

We design some classes and properties for the descriptive knowledge of the knowledge base, and design some reasoning rules for the inferential knowledge.

Now, we briefly introduce some ontology classes, some properties, and some individuals about data structures and algorithms. Their names and types are shown in TABLE I. The content of the column named Name are names of classes, properties and individuals, and the column named Type shows their types. In TABLE I, the names in the rows numbered 2, 4, 6, 8 and 11 are concerned with data structures, and they can be replaced by concrete names about their respective data structures. Similarly, the names in the rows numbered 3, 5, 7, 9 and 12 are concerned with algorithms, and they can be replaced by concrete names about their respective algorithms.

The domain and range of object properties and data properties are displayed in TABLE II. The domains of the object properties are the class SourceCode, their ranges are respectively the signal classes (e.g., DS_Signal_Class). The functions of these object

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SourceCode</td>
<td>Class</td>
</tr>
<tr>
<td>2</td>
<td>DS_Signal_Class</td>
<td>Class</td>
</tr>
<tr>
<td>3</td>
<td>AL_Signal_Class</td>
<td>Class</td>
</tr>
<tr>
<td>4</td>
<td>has_ds_signal_object</td>
<td>Object Property</td>
</tr>
<tr>
<td>5</td>
<td>has_al_signal_object</td>
<td>Object Property</td>
</tr>
<tr>
<td>6</td>
<td>has_ds_signal_data</td>
<td>Data Property</td>
</tr>
<tr>
<td>7</td>
<td>has_al_signal_data</td>
<td>Data Property</td>
</tr>
<tr>
<td>8</td>
<td>has_ds_used_data</td>
<td>Data Property</td>
</tr>
<tr>
<td>9</td>
<td>has_al_used_data</td>
<td>Data Property</td>
</tr>
<tr>
<td>10</td>
<td>Code_individual</td>
<td>Individual</td>
</tr>
<tr>
<td>11</td>
<td>DS_Signal_Class_individual</td>
<td>Individual</td>
</tr>
<tr>
<td>12</td>
<td>AL_Signal_Class_individual</td>
<td>Individual</td>
</tr>
</tbody>
</table>
properties are to mark whether or not a program code (indicated by an individual, e.g., Code_individual, of the class SourceCode) has contained some kinds of signals about the data structure DS and the algorithm AL (indicated by respective signal classes, e.g., DS_Signal_Class). The domains of the properties has_ds_signal_data and has_al_signal_data are their respective signal classes (e.g., DS_Signal_Class), and their range types are String. These data properties can store the specific signal strings which may appear in the program codes. The domains of the data properties has_ds_used_data and has_al_used_data are the class SourceCode, and their ranges are Boolean. The function of the data property has_ds_used_data is to indicate whether or not a program code uses the data structure ds. The function of the data property has_al_used_data is to indicate whether or not a program code uses the algorithm al.

C. Descriptive Knowledge about Algorithms

Here, we define some variables to denote the classes, properties and individuals about algorithms in the knowledge base as follows.

- \( A \) is the set of the algorithms in the knowledge base;
- \( A_i \) is an algorithm \( i \) in the set \( A \), e.g., \( A_i = DFS \);
- \( S(i, j) \) is one of ontology classes indicated by AL_Signal_Class in TABLE I, which represents a kind \( j \) of signals of the algorithm \( A_i \);
- \( I\text{Aset}(i, \ j, \ k) \) is one of ontology individuals indicated by AL_Signal_Class_individual in TABLE I, which is a member of the class \( S(i, j) \), representing a signal \( k \) belonging to a kind \( j \) of signals of the algorithm \( A_i \);
- \( DPS(i, j) \) is one of data properties indicated by has_al_signal_data in TABLE II, whose domain is the class \( S(i, j) \) and whose range is a set of strings representing a kind \( j \) of signals of the algorithm \( A_i \);

Take the algorithm DFS (Depth First Search) in Fig.2 for example, there is only one class DepthFirstSearch for its signals, i.e., \( S(i, 1) = \text{DepthFirstSearch} \). It has a data property has_depth_first_search_signal, i.e., \( DPS(i, 1) = \text{has_depth_first_search_signal} \). Furthermore, there are seven individuals in the class \( S(i, 1) \). The first individual is \( I\text{Aset}(i, 1, 1) = \text{DepthFirstSearch_individual_1} \), whose property value is “dfs”. The last individual is \( I\text{Aset}(i, 1, 7) = \text{DepthFirstSearch_individual_7} \), whose property value is “depth_first_search”.

D. Descriptive Knowledge about Data Structures

The variables needed to denote the classes, properties and individuals about data structures in the knowledge base are as follows.

- \( D \) is the set of the data structures in the knowledge base;
- \( D_i \) is a data structure \( i \) in the set \( D \), e.g., \( D_i = \text{Stack} \);
- \( Sd(i, j) \) is one of ontology classes indicated by DS_Signal_Class in TABLE I, which represents a kind \( j \) of signals of the data structure \( D_i \);
- \( IDset(i, j, k) \) is one of ontology individuals indicated by DS_Signal_Class_individual in TABLE I, which is a member of the class \( Sd(i, j) \), representing a signal \( k \) belonging to a kind \( j \) of signals of the data structure \( D_i \);
- \( DPSd(i, j) \) is one of data properties indicated by has_ds_signal_data in TABLE II, whose domain is the class \( Sd(i, j) \) and whose range is a set of strings representing the kind \( j \) of signals of the data structure \( D_i \).

Take the data structure \( \text{Stack} \) in Fig.3 for example, there are three classes StackVariable, StackPopOperation and StackPushOperation for its three kinds of signals, i.e., \( Sd(i, 1) = \text{StackVariable}, Sd(i, 2) = \text{StackPopOperation}, \) and \( Sd(i, 3) = \text{StackPushOperation} \). The class \( Sd(i, 1) \) has a data property has_stack_variable_signal, i.e., \( DPSd(i, 1) = \text{has_stack_variable_signal} \). There are two individuals in the class \( Sd(i, 1) \). The first individual is \( IDset(i, 1, 1) = \text{StackVariable_individual_1} \), whose property value is “stack”. The second individual is \( IDset(i, 1, 2) = \text{StackVariable_individual_2} \), whose property value is “mysta”. The class \( Sd(i, 2) \) has a data property has_stack_pop_operation_signal, i.e., \( DPSd(i, 2) = \text{has_stack_pop_operation_signal} \). There are two individuals in the class \( Sd(i, 2) \). The first individual is \( IDset(i, 2, 1) = \text{StackPopOperation_individual_1} \), whose property value is “pop”. The second individual is \( IDset(i, 2, 2) = \text{StackPopOperation_individual_2} \), whose property value is “pop_stack”. The class \( Sd(i, 3) \) has a data property has_stack_push_operation_signal, i.e., \( DPSd(i,
3) \( = \text{has\_stack\_push\_operation\_signal}\). There are two individuals in the class \( Sd(i, 3)\). One is \( Ds(i, 3, 1) = \text{StackPushOperation\_individual\_1}\), whose property value is "push"; whereas the other is \( Ds(i, 3, 2) = \text{StackPushOperation\_individual\_2}\), whose property value is "push\_stack".

Figure 3. The descriptive knowledge about the data structure Stack.

E. Descriptive Knowledge about Program Codes

We use SC to represent the ontology class SourceCode and its properties about the algorithm DFS and the data structure Stack. The class SC has an object property \( OPa(i, j) = \text{has\_depth\_first\_search}\), which corresponds to the only kind of signals about the algorithm DFS. Furthermore, it has another three object properties \( \text{has\_stack\_variable}\), \( \text{has\_stack\_pop\_operation}\) and \( \text{has\_stack\_push\_operation}\) which correspond to the three kinds of signals of the stack, respectively, i.e., \( OPd(i, 1) = \text{has\_stack\_variable}\), \( OPd(i, 2) = \text{has\_stack\_pop\_operation}\) and \( OPd(i, 3) = \text{has\_stack\_push\_operation}\). Finally, the class SC has two data properties, i.e., \( DPUs(i) = \text{has\_ds\_used}\) and \( DPUs(i) = \text{has\_stack\_used}\).

Fig.4 depicts the class SC (i.e., SourceCode) and its properties about the algorithm DFS and the data structure Stack. The class SC has an object property \( OPa(i, j) = \text{has\_depth\_first\_search}\), which corresponds to the only kind of signals about the algorithm DFS. Furthermore, it has another three object properties \( \text{has\_stack\_variable}\), \( \text{has\_stack\_pop\_operation}\) and \( \text{has\_stack\_push\_operation}\) which correspond to the three kinds of signals of the stack, respectively, i.e., \( OPd(i, 1) = \text{has\_stack\_variable}\), \( OPd(i, 2) = \text{has\_stack\_pop\_operation}\) and \( OPd(i, 3) = \text{has\_stack\_push\_operation}\). Finally, the class SC has two data properties, i.e., \( DPUs(i) = \text{has\_ds\_used}\) and \( DPUs(i) = \text{has\_stack\_used}\).

Figure 4. Some properties of the class SourceCode.

The procedure to create classes and properties is described briefly as follows.

First, we create the unified ontology class SC, and its properties \( DPUs(i)\) and \( DPUs(i)\).

Then, for each kind \( j\) of signals of each data structure \( i\) we build a class \( Sd(i, j)\) and its property \( DPDs(i, j)\), and the property \( OPd(i, j)\) of the class SC. For each kind \( j\) of signals of each algorithm \( i\) we also build a class \( Sa(i, j)\).
and its property \(DPSa(i, j)\), and the property \(OPa(i, j)\) of the class \(SC\).

Finally, for each signal \(k\) belonging to the kind \(j\) of signals of the data structure \(i\) we create a member \(ID\text{set}(i, j, k)\) of the class \(Sa(i, j)\), whose data property \(DPS\text{d}(i, j)\) is assigned with the string of the signal \(k\). Similarly, we create a member \(I\text{set}(i, j, k)\) of a class \(Sa(i, j)\) for each signal \(k\) of the algorithm \(i\), whose data property \(DPS\text{a}(i, j)\) is assigned with the string of the signal \(k\).

We regard these classes, properties and individuals as the descriptive knowledge in the knowledge base.

F. Inferential Knowledge for Reasoning Rules

The inferential knowledge in the knowledge base refers to the rules to recognize data structures or algorithms used in program codes. A rule usually consists of a conclusion and several preconditions. The conclusion is the reasoning result that we want to obtain, and the preconditions indicate the circumstances under which the conclusion is tenable. The conclusion is associated with one of properties indicated by \(has\_al\_used\_data\) or \(has\_ds\_used\_data\) in TABLE II, which would appear in the query for recognizing the corresponding data structure or algorithm. Each precondition is also associated with a respective property.

Fig.5 shows the reasoning rule of the data structure \(Stack\). In this rule, there are four properties marked by the tag \(<\text{ruleml:}\text{individualPropertyAtom}\>_0\). The property named \(has\_stack\_used\) between the tag \(<\text{ruleml:}\text{head}\>_0\) and the tag \(<\text{ruleml:}\text{head}\>_1\) is the conclusion of the rule. It has two variables with the same name \(CODE\). Its meaning is to check up whether or not an individual (indicated by \(CODE\)) of the class \(SC\), which represents a program code, satisfies the following preconditions. If it satisfies all the preconditions, the individual of the class \(SC\) will be added to the result set of the rule. The properties between the tag \(<\text{ruleml:}\text{body}\>_0\) and the tag \(<\text{ruleml:}\text{body}\>_1\) are three preconditions of the rule. Taking the property named \(has\_stack\_variable\) for example, it possesses two variables named \(CODE\) (which is the same as the variable \(CODE\) appearing in the conclusion) and \(STACKVAR\). It means that when the conclusion is tenable, the individual (indicated by \(CODE\)) of the class \(SC\) must have an object property \(has\_stack\_variable\). When the individual indicated by \(CODE\) possesses the three object properties \(has\_stack\_variable\), \(has\_stack\_pop\_operation\) and \(has\_stack\_push\_operation\) (which are part of knowledge depicted in Fig.4), this rule infers that the program code represented by the individual indicated by \(CODE\) use the data structure \(Stack\).

We regard the reasoning rules as the inferential knowledge of the knowledge base. For each data structure or each algorithm, there is a unique rule associated with it in the knowledge base. Thus, each rule represents a data structure or an algorithm in the knowledge base. The inferential knowledge in the knowledge base would be used to recognize the data structures or algorithms lying in program codes.

III. RELATIONSHIP DISCOVERY

A. Ontology Individuals for Program Codes

After building the knowledge base, we could create individuals of the class \(SC\) for program codes, and attach to the created individuals some properties if the program codes contain some corresponding signals about data structures or algorithms in the knowledge base.

We propose an algorithm \(CreateIndividual\), which is used to create an individual for a program code. The following variables are needed in this algorithm.

- \(C\) is the program code;
- \(Ind(C)\) is the individual created for the program code \(C\).

The algorithm \(CreateIndividual\) is described as follows:

1. Read the content of a program code \(C\);
2. Create an ontology individual \(Ind(C)\) of the class \(SC\) for the program code \(C\) (In this step, the individual \(Ind(C)\) does not possess any property);
3. For each individual \(ID\text{set}(i, j, k)\) in each signal class \(Sa(i, j)\) of each data structure \(D\), get the signal string \(k\) by reading the value of its data property \(DPS\text{d}(i, j)\), and attach the individual \(ID\text{set}(i, j, k)\) to the code individual \(Ind(C)\) via the object property \(OPd(i, j)\) if the program code \(C\) contains the signal string \(k\);
4. For each individual \(I\text{set}(i, j, k)\) in each signal class \(Sa(i, j)\) of each algorithm \(A\), get the signal string \(k\) by reading the value of its data property \(DPS\text{a}(i, j)\), and attach the individual \(I\text{set}(i, j, k)\) to the code individual \(Ind(C)\) via the object property \(OPa(i, j)\) if the program code \(C\) contains the signal string \(k\).

Fig.6 shows a program code which contains signals of three kinds of the data structure \(Stack\), and the signal string “dfs” of the algorithm \(DFS\). By using the algorithm \(CreateIndividual\), we create a code individual named \(ProgramCode_1\) for this program code as shown in Fig.7.

© 2013 ACADEMY PUBLISHER
/*\include<stdio.h>
#include<stack>
#define maxn 50
using namespace std;
stack < int > mysta;
int lena, lenb, num[maxn];
char a[maxn], b[maxn];
void dfs(int i, int j, int k) {
    char ch; int p;
    if (j == lenb) {
        printf("\n");
        return;
    }
    if (num[p]) {
        for (p = 0; p < k; p++) {
            if (num[p]) {
                printf("i ");
            } else {
                printf("o ");
            }
            printf("\n");
            return;
        }
    }
    printf("\n");
    printf("z");
    if (i < lena) {
        mysta.push(a[i]);
        num[k] = 1;
        dfs(i + 1, j, k + 1);
        mysta.pop();
    }
    if (mysta.empty() &&
        mysta.top() == b[j]) {
        ch = mysta.top();
        mysta.pop();
        num[k] = 0;
        dfs(i, j + 1, k + 1);
        mysta.push(ch);
    }
}
int main() {
    scanf("%s%s", a, b);
    while (scanf("%s", a, b) != EOF) {
        lena = strlen(a);
        lenb = strlen(b);
        while (!(mysta.empty() &&
            mysta.top() == b[j])) {
            ch = mysta.top();
            while (scanf("%s", a, b) != EOF) {
                lena = strlen(a);
                lenb = strlen(b);
                while (!(mysta.empty() &&
                    mysta.top() == b[j])) {
                    if (mysta.empty() &&
                        mysta.top() == b[j]) {
                        ch = mysta.top();
                        mysta.pop();
                        num[k] = 0;
                        dfs(i, j + 1, k + 1);
                        mysta.push(ch);
                    }
                }
            }
        }
        printf("\n");
    }
    return;
}

Figure 6. A program code which employs DFS and Stack.

B. Recognition of Programming Knowledge

After creating individuals for program codes, we are now ready to recognize the programming knowledge of data structures or algorithms lying in the program codes by using the inferential knowledge in the knowledge base.

In order to recognize the data structures or the algorithms, we use Java programming language to create an instance of a KAON2 [14] reasoner.

For each rule in the knowledge base, a query is built to recognize its corresponding data structure or algorithm. The query is used to search for all individuals of the class SourceCode that satisfy the preconditions of the reasoning rule. The result of the query is a group of individuals in the class SourceCode which satisfies the preconditions and therefore indicates that its corresponding program code contains the data structure (or algorithm) associated with the reasoning rule.

Fig. 8 shows a query about the data structure Stack. The literal hasStackDS_CODE_STACK uses the property has_stack_used, which is the rule for the data structure Stack in Fig. 5. This literal is used by the query codeContainsDataStructure, so that the query about Stack could be associated with the rule of Stack. The query uses another literal for the class SourceCode, which indicates that the scope of the query is the individuals of the class SourceCode.

When the query, whose conditions are as the rule defines, is open, the reasoning process would be executed. As a result, the individuals which possess the three object properties has_stack_variable, has_stack_push_operation and has_stack_pop_operation would be selected.

We define some variables as follows.

- $Q_i$ indicates the query for a data structure $i$ in the knowledge base;
- $R$ indicates an instance of the KAON2 reasoner, which could execute the queries about the data structures and algorithms;
- $C_i$ indicates the result of the query $Q_i$, which is a set of individuals in the class $SC$ that satisfy the preconditions of the reasoning rule.

The algorithm, which is used to recognize a data structure $i$, is described as follows.

1. For each individual $Ind(C_i)$ in the knowledge base, attach to it a data property $DP_Ui(i)$ whose value is assigned with $false$;
2. Create an instance $R$;
3. Build a query $Q_i$ on a basis of the rule about the data structure $i$ in the knowledge base;
4. Get the result set $C_i$ by executing the query $Q_i$;
5. For each individual in the set $C_i$, set $true$ to its data property $DP_Ui(i)$.

For each data structure in the knowledge base, do the above steps to recognize the data structure, and attach its corresponding property to each individual $Ind(C_i)$ of program codes. The steps used to recognize algorithms are similar to the steps of the above algorithm. Fig. 9 depicts the individual of the program code in Fig. 6, whose data properties has_stack_used and has_dfs_used are assigned with $true$. The recognition results would be

\begin{verbatim}
<:SourceCode rdf:ID="ProgramCode_1"><has_stack_variable rdf:resource="#StackVariable_individual_2"/>
<has_stack_pop_operation rdf:resource="#StackPopOperation_individual_1"/>
<has_stack_push_operation rdf:resource="#StackPushOperation_individual_1"/>
<has_depth_first_search rdf:resource="#DepthFirstSearch_individual_1"/>
</:SourceCode>

Figure 7. The code individual for the program code.

\begin{verbatim}
<:SourceCode rdf:ID="ProgramCode_1"><has_stack_variable rdf:resource="#StackVariable_individual_2"/>
<has_stack_pop_operation rdf:resource="#StackPopOperation_individual_1"/>
<has_stack_push_operation rdf:resource="#StackPushOperation_individual_1"/>
<has_depth_first_search rdf:resource="#DepthFirstSearch_individual_1"/>
</:SourceCode>

Figure 8. The query about Stack in Java language.

Figure 9. The code individual with recognition result.
\end{verbatim}
used to discover the relationships between data structures and algorithms.

C. Discovery of Programming Knowledge Relationships

We define that the relationships between data structures and algorithms are reflected by the frequencies of the two kinds of programming knowledge used in program codes simultaneously. When a data structure and an algorithm are contained in a program code, there may be a relationship between them. For instance, the data structure Stack and the algorithm Depth First Search are often contained in a program code, which implies a relationship between them.

We propose a 3-tuple data model \( R = <A, D, N> \) to mark the relationships between algorithms and data structures lying in the program codes. Here, \( R \) denotes the set of relationships, \( A \) denotes the set of algorithms in the knowledge base, \( D \) denotes the set of data structures in the knowledge base and \( N \) denotes the set of frequencies. We assume that there are \( m \) algorithms and \( n \) data structures, and there are \( m \times n \) different relationships between algorithms and data structures. Thus, each of the 3-tuples can be expressed as \( R_{i,j} = <A_i, D_j, N_{i,j}> \). In this tuple, \( A_i \) \((i = 1, 2, \ldots, m) \) denotes an algorithm in the set \( A \); \( D_j \) \((j = 1, 2, \ldots, n) \) denotes a data structure in the set \( D \); \( N_{i,j} \) \((i = 1, 2, \ldots, m, j = 1, 2, \ldots, n) \) shows the frequency of the algorithm \( A_i \) and the data structure \( D_j \) used simultaneously in the same program codes.

The value of each element \( N_{i,j} \) is the number of all program codes that use both the algorithm \( A_i \) \((i = 1, 2, \ldots, m) \) and the data structure \( D_j \) \((j = 1, 2, \ldots, n) \). This value can be obtained by counting the number of individuals whose data properties \( DP{Uu}(i) \) and \( DP{Ud}(j) \) are both true. Thus, the relationship \( R_{i,j} \) between an algorithm \( A_i \) and a data structure \( D_j \) is indicated by the 3-tuple \( <A_i, D_j, N_{i,j}> \).

According to the elements of the set \( R \), we can calculate the total frequency \( A_{i\text{Total}} \) of each algorithm \( A_i \) by the formula in (1), where \( A_{i\text{Total}} \) means the total number of the program codes that use the algorithm \( A_i \) simultaneously with at least one data structure in the knowledge base.

\[
A_{i\text{Total}} = \sum_{j=1}^{n} N_{i,j} 
\]

For each pair of a data structure \( D_j \) and an algorithm \( A_i \), we give a definition that the rate \( Rate_{i,j} \) of the data structure \( D_j \) is the ratio of the number \( N_{i,j} \) to the number \( A_{i\text{Total}} \). The rate \( Rate_{i,j} \) indicates how much the data structure \( D_j \) is related to the algorithm \( A_i \). Its value is between 0 and 1 inclusive. The bigger the value, the more related to the algorithm \( A_i \) the data structure \( D_j \) is. The rate can be calculated by the formula in (2).

\[
Rate_{i,j} = \frac{N_{i,j}}{A_{i\text{Total}}} 
\]

IV. LEARNING METRIALS

In this Section, we will introduce the programming problems on OJ systems, and then organize the programming problems and their program codes on the web into learning materials in terms of algorithms and data structures with hyperlink techniques.

A. Information Items of Programming Problems

A programming problem from an OJ system may consist of many information items. We just consider the following items: Pro.ID, problem name, problem URL, problem description, and pass rate.

We define that Pro.ID consists of two parts, the name of the OJ system and the serial number of a programming problem on the OJ system.

The pass rate of a programming problem is defined as the percentage of the accepted program codes out of all the program codes submitted by students for the sake of solving the programming problem. The pass rate of a programming problem reflects its difficulty. If the pass rate of a programming problem is low, it means that the problem is difficult.

The web page of a programming problem contains some information items about this problem.

Fig.10 shows a part of the web page of a programming problem from the HDU \(^1\) system. We could get the information items about this problem from this page as shown in TABLE III. This programming problem has its serial number “3078” on the OJ system named “HDU”, so its Pro.ID is “HDU 3078”. From this page, we can see that its problem name is “Network”. The description of the programming problem is the string “The ALPC company is.....”. The total number of program codes submitted by students for the programming problem is 96, and the number of the accepted codes is 54. Thus, the pass rate of this programming problem is 56.25%, which

\(^1\) HDU: http://acm.hdu.edu.cn/
is the number of accepted codes divided by the total number of program codes.

B. Obtainment of Information Items

The Pro.ID and the name of a programming problem can be obtained during the process of downloading solution reports from the web if we use the method mentioned in [1].

According to the Pro.ID of a programming problem, we can get the URL of the problem by string concatenation. TABLE IV shows the URLs of programming problems on three different OJ systems, where “xxxx” is the problem serial number. Take the programming problem in TABLE III for example, its Pro.ID is “HDU 3078”, so that its problem URL is “http://acm.hdu.edu.cn/showproblem.php?pid=3078”. By using this URL, we can obtain the web page which contains its name and description as shown in Fig.10.

The pass rate of a problem is contained in the volume page which this problem belongs to. We can get its pass rate from the volume page.

C. Organization of program codes and problems

We download solution reports on the web for programming problems using the method described in [1]. During the process, the URLs of reports, Pro.ID and problem names could be obtained. According to Pro.ID, we can get the URL of a problem, and then obtain its pass rate.

We get program codes from solution reports, and then discover the relationships between algorithms and data structures used in them.

With the discovered relationship, we could organize the program codes on the web and the programming problems on online judges into learning materials in terms of algorithms and data structures.

D. Structure of Learning Metrials

The learning materials possess a hierarchical structure. The highest level of this structure is the algorithms. Below each algorithm, there are several data structures, which are sorted with their percents from high to low. For each data structure, several problems, which are sorted by their pass rates from high to low, are displayed. The materials also provide one or more solution reports for each programming problem.

So these materials can help learners to study programming knowledge in terms of algorithms and data structures systematically.

V. EXPERIMENT AND RESULT

In our experiment, we took program codes written in C/C++ programming language and the programming problems from OJ systems as our research data. The program codes were obtained from solution reports, which were downloaded from many web sites, and the programming problems were from three OJ systems HDU, POJ², and ZOJ³.

We built a knowledge base for seven algorithms and twelve data structures by using the ontology tool protégé [15]. The seven algorithms are BFS, DFS, MaxFlow, ShortestPath, MST, Topological and Linear DP. The twelve data structures are Stack, Queue, LinkList, UnionFindSet and the eight data structures from the Standard Template Library (STL).

We collected 6643 solution reports of 480 programming problems, and obtained 3752 program codes written in the C/C++ programming language from these reports. We created 3752 individuals of the class SourceCode for the 3752 program codes respectively, and recognized data structures and algorithms used in these program codes.

Since some of the program codes did not use any data structure or any algorithm in the knowledge base we created, for the sake of discovering the associated relationships between data structures and algorithms, we selected 1073 suitable program codes, which use both data structures and algorithms according to the recognition results.

Some results of our experiment are shown in TABLE V. In the string “HDU_2101_code_1” from the first column, “HDU” is the OJ system name, “2101” is the serial number of the problem on the HDU system, and “code_1” means that this code is from the first solution report of the programming problem with the Pro.ID “HDU 2101”. The second column is the recognition result of data structures and the last one is the recognition result of algorithms.

After recognizing the data structures and algorithms used in the program codes, we can analyze the relationships between the two kinds of knowledge. We calculated the total frequency $A_{Total}$ of each algorithm $A_i$ by the formula in (1), and the rate $Rate_{A_i,D_j}$ of each data structure $D_j$ which is used simultaneously with the algorithm $A_i$ by the formula in (2).

<table>
<thead>
<tr>
<th>OJ Name</th>
<th>URL of Problem Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDU</td>
<td><a href="http://acm.hdu.edu.cn/showproblem.php?pid=xxxx">http://acm.hdu.edu.cn/showproblem.php?pid=xxxx</a></td>
<td>HDU.com</td>
</tr>
<tr>
<td>POJ</td>
<td><a href="http://poj.org/problem?id=xxxx">http://poj.org/problem?id=xxxx</a></td>
<td>POJ.org</td>
</tr>
<tr>
<td>ZOJ</td>
<td><a href="http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=xxxx">http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=xxxx</a></td>
<td>ZOJ.com</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Code Name</th>
<th>Data Structure</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDU_2101_code_1</td>
<td>STLQueue</td>
<td>BFS</td>
</tr>
<tr>
<td>POJ_3522_code_1</td>
<td>UnionFindSet</td>
<td>MST</td>
</tr>
<tr>
<td>POJ_2337_code_1</td>
<td>STL Stack</td>
<td>DFS</td>
</tr>
<tr>
<td>POJ_3272_code_1</td>
<td>STLQueue</td>
<td>Topological</td>
</tr>
</tbody>
</table>
TABLE VI shows the relationships between the seven algorithms and several data structures used in program codes simultaneously. The second column lists the algorithm names, each followed by the number (in the parentheses) of all program codes that use the corresponding algorithm. The third column displays the data structures, which are used in program codes simultaneously with the corresponding algorithms. The last column shows the rate Rate(i,j) of the data structure Dj related to the algorithm Ai. The data structures, which are used in program codes simultaneously with the same algorithm, are sorted by their rate Rate(i,j) from high to low.

<table>
<thead>
<tr>
<th>No</th>
<th>Ai (AiTotal)</th>
<th>Dj</th>
<th>Rate(i,j)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BFS (650)</td>
<td>STL Queue</td>
<td>70.17%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Priority Queue</td>
<td>12.52%</td>
</tr>
<tr>
<td>2</td>
<td>DFS (371)</td>
<td>STL Queue</td>
<td>27.49%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Vector</td>
<td>24.53%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Stack</td>
<td>10.24%</td>
</tr>
<tr>
<td>3</td>
<td>MaxFlow (77)</td>
<td>STL Queue</td>
<td>83.12%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Priority Queue</td>
<td>62.16%</td>
</tr>
<tr>
<td>4</td>
<td>ShortestPath (74)</td>
<td>STL Queue</td>
<td>13.51%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Vector</td>
<td>10.81%</td>
</tr>
<tr>
<td>5</td>
<td>MST (57)</td>
<td>UnionFindSet</td>
<td>42.11%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Queue</td>
<td>24.56%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Priority Queue</td>
<td>7.02%</td>
</tr>
<tr>
<td>6</td>
<td>Topological (32)</td>
<td>STL Queue</td>
<td>40.63%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Priority Queue</td>
<td>18.75%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Stack</td>
<td>12.50%</td>
</tr>
<tr>
<td>7</td>
<td>Linear DP (24)</td>
<td>STL Vector</td>
<td>33.33%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Queue</td>
<td>33.33%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STL Map</td>
<td>12.50%</td>
</tr>
</tbody>
</table>

Fig.11 shows the hierarchical structure of programming knowledge in terms of algorithms and data structures. At the top level are algorithm names and at the lower level are data structure names. For each algorithm, it displays its correlative data structures which are sorted by the percentage from high to low. Take the algorithm DFS as an example, the number 371 in the parentheses which follows the algorithm name means that there are 371 program codes out of 1073 program codes use this simultaneously with DFS, are STL Queue (102), STL Vector (91) and STL Stack (38).

VI. CONCLUSION

We have proposed an approach to discover the relationships between data structures and algorithms with ontology techniques, which consists of constructing the knowledge base about data structures and algorithms, recognizing the two kinds of programming knowledge used in program codes with ontology reasoning technique, and discovering the relationships between them by recording their frequencies. With the discovered relationships, we organized the program codes on the web and the programming problems distributed on OJ systems into learning materials with a hierarchical structure. In this structure, the top level is algorithms, and the lower is data structures. Under the level of data structures, several problems are provided in the descending order of their pass rates. For each problem, we give one or more solution reports for students to study. Fig.12 shows two programming problems whose solutions use both DFS and STL Stack. The Pro.IDs of them are “HDU 3078” and “POJ 3114”, respectively. The pass rate of the problem named “Network” is 56.26%, and it is higher than the pass rate 31.87% of the problem named “Countries in War”, thus the former problem is displayed above the latter one.
ACKNOWLEDGMENT

This research is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 60973121.

REFERENCES


Guojin Zhu is an associate professor at the Department of Computer Science, Donghua University (DHU), Shanghai, China. He received his M.S. and Ph.D. degrees from DHU in 1991 and 2007, respectively. He was a visiting scholar at the Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, USA from November 2007 to November 2008. His current research interests include semantic web, knowledge discovery, and neural computing.

Zhiyue Yu is a graduate student of Computer Application Technology at Donghua University. She was born in Hebei province, P.R. China in 1987, and received her Bachelor of Science degree from Shaanxi University of Science and Technology in 2010. Her current main research interest is computer network and artificial intelligence.

Jiyun Li is an associate professor at the Department of Computer Science, Donghua University (DHU), Shanghai, China. She received her M.S. and Ph.D. degrees from DHU in 1996 and 2003, respectively. She has been visiting scholar at the Department of Psychology and Brain Science in Indiana University and Textile and Fashion Institute in Hongkong Polytechnique University. Her current research interests include cognitive modeling, knowledge discovery, and neural computing.